Towards Advanced and Interactive Web Perspective View Services

Benjamin Hagedorn, Dieter Hildebrandt, and Jürgen Döllner

Hasso-Plattner-Institute at the University of Potsdam,
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany,
{benjamin.hagedorn|dieter.hildebrandt|doellner}@hpi.uni-potsdam.de

Abstract. The Web Perspective View Service (WPVS) generates 2D images of perspective views of 3D geovirtual environments (e.g., virtual 3D city models) and represents one fundamental class of portrayal services. As key advantage, this image-based approach can be deployed across arbitrary networks due to server-side 3D rendering and 3D model management. However, restricted visualization and interaction capabilities of WPVS-based applications represent its main weaknesses. To overcome these limitations, we present the concept and an implementation of the *WPVS++*, a WPVS extension, which provides A) additional thematic information layers for generated images and B) additional service operations for requesting spatial and thematic information. Based on these functional extensions, *WPVS++* clients can implement various 3D visualization and interaction features without changing the underlying working principle, which leads to an increased degree of interactivity and is demonstrated by prototypic web-based client applications.

1 Introduction

Within spatial data infrastructures, geovisualization plays an important role as it allows humans to understand complex spatial settings and to integrate heterogeneous geodata from distributed sources on the visualization level. For this purpose, the Open Geospatial Consortium (OGC) as a standardization organization proposes several portrayal services for 2D and 3D geodata. So far, there is only one widely used "workhorse", the Web Map Service (WMS) [4], for generating 2D maps. Standards for visualizing 3D geovirtual environments (3DGeoVEs) such as virtual 3D city models and

landscape models have not been elaborated to a similar degree. For portraying 3D geodata, two approaches have been presented: the Web 3D Service (W3DS) and the Web Perspective View Service (WPVS).[1]

While the W3DS provides scene graphs that have to be rendered by the service consumer, the WPVS supports the generation of images that show 2D perspective views of 3DGeoVEs encoded in standard image formats. The key advantages of this image-based approach include low hardware and software requirements on the service consumer side due to service-side 3D model management and 3D rendering, and moderate bandwidth requirements; only images need to be transferred regardless of the model and rendering complexity. The weaknesses of the current WPVS approach include the limited degree of interactivity of client applications. For example, it is not possible to navigate to a position identified in the image or to retrieve information about visible geographic objects (e.g., buildings).

To overcome these limitations, we propose the *WPVS++*, an extension of the OGC WPVS that is capable of augmenting each generated color image by multiple thematic information layers encoded as images. These additional image layers provide for instance depth information and object identity information for each pixel of the color image. Additionally, we propose operations for retrieving thematic information about presented objects at a specific image position, simple measurement functionality, and enhanced navigation support. This allows *WPVS++* clients to efficiently access information about visually encoded objects in images, to use that information for advanced and assistant 3D navigation techniques, and thereby to increase the degree of interactivity, which is demonstrated by prototype implementations of two web-based clients.

The remainder of this paper is organized as follows. Section 2 describes and analysis approaches for service-based 3D portrayal and reviews related work. Section 3 presents the extended information model of the *WPVS++*; the extended interaction model is proposed in Section 4. Section 5 describes our prototype implementation. Section 6 discusses results, and Section 7 concludes the paper.

[1] The W3DS has discussion status. The WPVS is an OGC-internal draft specification and represents the successor of the OGC Web Terrain Server discussion paper [15].

2 Background and Related Work

2.1 Service-Based 3D Portrayal

The OGC portrayal model [12] allows for three principle approaches for distributing the tasks of the general rendering pipeline between portrayal services and consuming applications [1]. These segmentations differ in the type of exchanged data: either filtered geodata, or computer graphical representations, or finally rendered images are provided. This leads to different rendering capabilities and hardware and software requirements of the portrayal services and the corresponding client applications.

The *OGC Web 3D Service* (W3DS) [13] generates 3D computer graphical representations, which are typically arranged as *scene graphs* and are transmitted to the service consumer using 3D graphics formats (e.g., VRML, X3D, COLLADA). This graphics data can include appearance information, e.g., surface colors, but is not capable of carrying thematic information. These representations have to be processed and rendered at the client side. Using the W3DS together with a powerful rendering client allows for real-time navigation and interaction in the 3D scene. However, for the provision of complex scenes by W3DS a high bandwidth is needed, as, e.g., massive geometry and texture data must be transferred.

In contrast, the *OGC Web Perspective View Service* (WPVS) is a 3D portrayal service for generating and providing finally rendered *images* of perspective views of a 3D scene. A WPVS can provide high-quality visualization, regardless of a client's rendering capabilities, limited only by the rendering engine of the server. As only standard images are transferred, the WPVS can be used by more simple clients and in situations where available bandwidth is low. Additionally, server-side 3D rendering implies that we do not rely on a possible driver, outdated or even unknown client-side 3D hardware, i.e., we do not have to handle client hardware configurations. Using a WPVS, 2D views of complex 3DGeoVEs can be specified by simple URLs and can be easily integrated into various applications and systems such as web portals. Navigation in a base WPVS is step-by-step and far from real-time interaction. We argue that this drawback can be attenuated by intelligent loading and display strategies (e.g., image preloading).

A major difference between WPVS and W3DS is in the application of optimization strategies for rendering. In the case of W3DS-based portrayal, optimization strategies such as tiling, culling, caching, and level-of-detail mechanisms are implemented at the client side, which is in charge of se-

lecting the right data to load from the W3DS. So, the W3DS client builds up a local context that is used for rendering. In contrast, with a WPVS optimization has to be implemented fully by the service, which does not provide any session management functionality. Thus, the WPVS rendering system cannot make any assumptions about the data to load and render. In worst case, an underlying (naive) rendering system would have to load and render different data for each single WPVS request.

2.2 Remote Visualization

Remote visualization comprises various approaches that distribute rendering functionalities within a network. In such systems, the data to visualize is processed remotely and transferred to a consumer as finally rendered image or video, or in a form that can be processed and rendered by the consumer more easily. Besides the OGC portrayal initiative, multiple other projects tackle the field of remote 3D visualization [9, 10].

At the client side, rendering techniques can be differentiated according to the type of data that is used as input for the rendering stage. They include, e.g., polygon-based rendering, point-based rendering, and image-based modeling and rendering. While both polygon-based and point-based rendering rely on geometry data, image-based rendering utilizes image information for deriving new views [3]. Image-based modeling includes techniques for reconstructing geometric information from images [5].

Mobile 3D visualization is mainly based on the programmable 3D graphics hardware that became available for mobile devices during the last years. It supports rendering 3D data directly on the mobile client. Here, a major challenge is to transfer geometry and texture data to the device. For this, elaborated data formats and rendering techniques for texture compression and buffer compression, frame buffer tiling, and geometry culling are required to reduce processing load and minimize power consumption [2].

2.3 Image Data and Formats

For rendering high-quality images, computer graphics hardware generates and takes into account per-vertex and per-fragment information such as vertex normal directions, texture coordinates, or depth values. This data is also incorporated by various rendering techniques for implementing specialized visual effects, e.g., object highlighting, edge enhancement, or ghost views. In computer graphics, the G-buffer [14] describes a concept for retrieving and storing this additional image information, which can be referred to as additional (i.e., non-color) image layers.

For enabling various visualization effects and techniques in post-processing, formats for storing these image layers are required. Especially for depth images, various work addresses data representation and transfer [7, 16]. OpenEXR [11] represents a format that supports several image channels in one image file, e.g., RGBA color values, luminance channels, depth values, and surface normals; it was originally designed for storing and editing high-dynamic-range (HDR) images.

Besides storing image layers as, e.g., arrays of float values, they can be encoded as color images. Storing data as images allows for utilizing standard image formats and their capabilities for data compression, which is crucial due to minimizing bandwidth usage when transferring image data over networks. Typical image compression algorithms include lossless Huffman encoding and Deflate. In [6], a technique that utilizes images as an efficient means for encoding geometry is presented. For each image layer type an encoding might exist that is more efficient than standard image encodings. However, in a standardization proposal in the context of service-oriented architectures, we rate more important the improved interoperability and reduced efforts for service consumers when using standard image encodings.

Various image formats allow for specifying the degree of compression, which leads to smaller data size, but possibly longer compression time and reduced image quality. Further, while lossy compression is acceptable for color values, it might be not for other image layers such as depth data.

3 Extending the Information Dimension

In this Section, we propose a concept for enhancing the WPVS by providing "semantic-rich" images, which complement the color images and serve as a basis for enhanced interaction and navigation.

3.1 Image Layers

Besides color images, we propose that the WPVS generates additional image layers storing information such as depth or object id values [14] for each pixel of the image. Fig. 1 shows examples of such image layers. We provide this additional data as separate images, in which each pixel value does not necessarily encode a color and is not necessarily dedicated for human cognition.

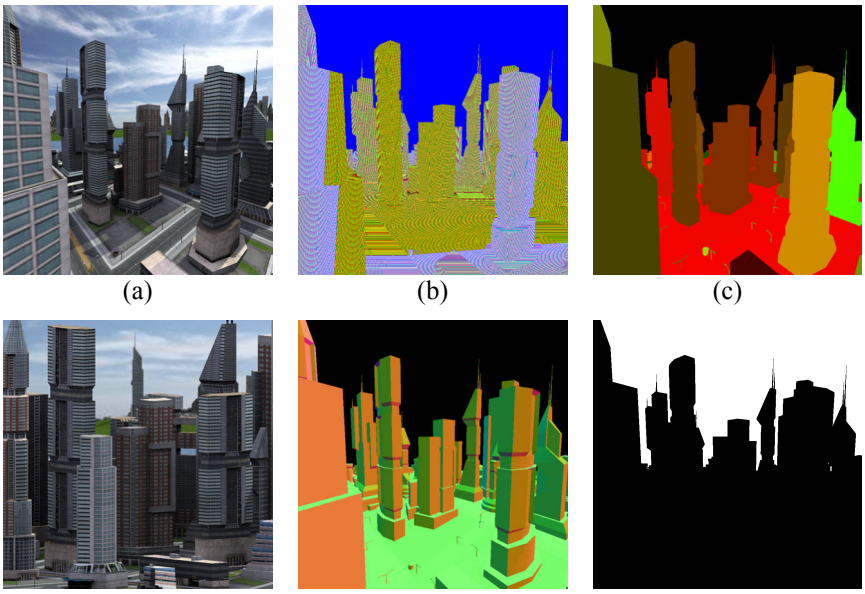

Fig. 1. Examples of image layers that are provided by *WPVS++*: color layer (a), depth layer (b), object id layer (c), normal layer (e), and mask layer (f). (d) shows an orthographic projection for the same location of the camera and point-of-interest.

Color Layer: A *color layer* contains a color value (e.g., RGB) for each pixel. According to the current WPVS specification, the service consumer can request a transparent background (RGBA), which requires image formats that support transparency, e.g., PNG or GIF.

Depth Layer: A *depth layer* describes the distance to the camera for each pixel. Depth images can serve for multiple effects, e.g., for computing 3D points at each pixel of the image (image-based modeling) or for various rendering techniques such as depth-of-field effects. Furthermore, depth information represents a major means to compose multiple images generated from the same camera position.

Typically, graphics hardware provides logarithmic and linearized depth values $z \in [0, 1]$. Different from this, we suggest providing depth values as distances to the virtual camera in world coordinates that correspond to the metrics defined by the requested *coordinate reference system* (CRS). This representation abstracts from computer graphical details and the distance values can be used without additional computation in many applications.

For this, each vertex p is transformed into camera space, and its linearized and normalized depth to the camera z_l is computed from its z-value p_z, and near plane *near* and far plane *far*. The graphics hardware interpolates the depth values for each fragment of an image; the value 0.0 is reserved for no-geometry fragments. In a second step for each image pixel, the depth value *depth* is computed in world metrics:

$$z_l = \frac{p_z - near}{far - near} \tag{1}$$

$$depth = near + z_l * (far - near) \tag{2}$$

Due to linearization, the precision of the depth values is decreasing for higher distances to the camera. For not reducing precision additionally, high bit rates are required for storing depth values.

As a first approach, we store depth values in IEEE 32 bit Float representation as images by segmenting their byte representations and assigning them to the color components of a pixel. Due to the direct storage as color components, resulting depth images possess very little pixel-to-pixel coherence, and should not be stored by lossy image formats. Thus, we suggest at least 32 bit (RGBA) images for this type of depth value storage.

Consequently, a service consumer has to recompose the depth values from the color components of the requested depth layer.

Object Id Layer: An *object id layer* contains a unique id for each pixel that refers to a scene object. Using this information, we can select all pixels that show a specific feature, e.g., for highlighting, contouring, or spatially analyzing the feature. For facilitating consistent interaction across multiple images, object ids should be unique over multiple requests. For that, they could be computed from an object id that is unique for the whole dataset.

Other Image Layers: In addition to the color, depth, and object id layers, we identified the following image layers to be relevant for the WPVS:

- *Normal Layer:* Describes the direction of the surface normal at each pixel. This could be used for the subsequent integration of additional light sources and the adjustment of the color values around that pixel, e.g., for highlighting scene objects. As a first approach, we encode normalized normal vectors in camera space by encoding each vector component (x, y, z) as color component (R, G, B) of a 24 bit color image.
- *Mask Layer:* Contains a value of 1 for each pixel that covers a scene object and 0 otherwise. Thus, a mask layer can be stored as 1 bit black and

white image. It could support the easy altering of the scene background or provide information about unused image space, which could be used, e.g., for blending advertisement, labels, etc.

- *Position Layer:* Contains the 3D coordinate at each 2D pixel. This allows the client to directly retrieve geo-position information, which, e.g., can be used for labeling or measurements.
- *Category Layer:* Classifies the image contents on a per-pixel basis; the classification could be based on, e.g., feature types, object usages, or object names. The category layer extends the so far described concept; it consists of an object id layer and a look-up table that lists for each object id the corresponding category (Fig. 2).

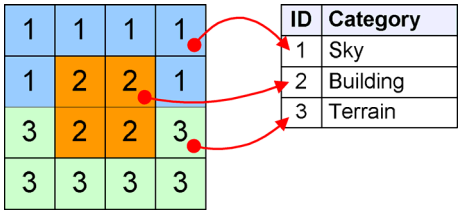

Fig. 2. A category layer consists of an object id layer and a category look-up table. Here, the pixels are categorized according to the feature type.

Image Data Encoding: Encoding also non-color data by standard image formats means to support the same principles for data encoding, data exchange, and client-side data loading and processing for all image data. Additionally, using images allows for applying state-of-the-art compression algorithms. For color data, which is dedicated for perception by humans, lossy compression algorithms can be used; for this data, JPEG mostly represents a suitable data encoding format. For data that requires exact values for each pixel, lossless image compression must be applied. We suggest PNG to encode this image data. PNG is also used for color images that shall contain transparency.

Technically we encode JPEG images by 24 bit per pixel and PNG images by 24 or 32 bit per pixel. Higher data accuracies could be reached by using image formats with higher bit rates, e.g., 32 (48) bit images or even 48 (64) bit images. Drawbacks include increased image size and, thus, transfer load, and processing time at both the service-side and the consumer-side.

3.2 Image Sets

Various applications could require to fetch multiple, spatially coherent image layers from the *WPVS++*. As the WPVS is a stateless service (i.e., each service request is self-contained and its processing is independent from any preceding request), it potentially has to fetch, map, and render disjunctive sets of geodata for each service request. In particular, this is relevant when considering large 3DGeoVEs or remote data sources to be visualized.

From a performance point of view, it is better to support multiple image layers within a single *WPVS++* response. Provided that the requested views are spatially coherent (i.e., they are located close to each or have overlapping view frustums), this could reduce the number of switches of the rendering context. As an additional benefit, the network communication overhead caused by multiple requests is reduced.

These image sets could provide multiple images layers for the same camera specification (e.g., a set consisting of a color layer, depth layer, and object id layer) as well as images for different camera specifications (e.g., perspective views along a path through the 3D world).

3.3 Convenient Camera Specification

The current OGC WPVS provides a camera specification that is based on a point-of-interest (POI) and is not convenient for many applications. For example, specifying a setting that shows a view out of a window to a specific point in the 3D world demands for laborious computations for deriving distance, pitch, yaw, and roll angles.

As various applications could profit from a different specification of the view frustum, we suggest replacing the camera specification by another one that is based on only three vectors: the 3D coordinate of the camera position (`POC`), the 3D coordinate of the point-of-interest (`POI`), and an optional camera up vector (`UP`) for specifying camera rolls (List. 1). Furthermore, parameters for near plane and far plane are required; they describe the culling volume used during the rendering process and are necessary, e.g., for the generation and usage of depth image layers. For the specification of distorted images, we suggest replacing the angle-of-view parameter by a field-of-view angle in degrees in horizontal direction (`FOVY`) and to complement it by the optional vertical angle (`FOVX`). If `FOVX` is not specified, it is derived from the image dimensions.

3.4 Additional Projection Types

The current OGC WPVS is intended for central perspective projections only. Complementary, an image-based 3D portrayal service could offer additional projection types, such as orthographic projections, which are used, e.g., for architectural applications.

For extending the *WPVS++* for orthographic projections, we suggest an alternative camera specification `Orthographic` that is controlled by the parameters `Left`, `Right`, `Top`, and `Bottom`, which describe the borders of the cuboid view frustum (List. 1).

Beyond perspective and orthographic projections, a 3D portrayal service could support more advanced projection types such as panoramic views (i.e., field-of-view larger than 180 degrees) or multi-perspective views [8].[2] While some of these projections could be generated from 2D images as provided by the current WPVS implementation, others require geometric information and, thus, are an integral part of the rendering process and need to be implemented by the 3D portrayal service itself.

3.5 Extended `GetView` Operation

A *WPVS++* `GetView` request mainly specifies the datasets to include, visualization styles to apply, and a camera specification; this information can be called image context. Adding the concept of multiple image layers, a *WPVS++* image context could be specified as described in List. 1.

Style specifications within this image context are dedicated and applicable only for requested color images. Parameters for the requested image layers allow for specifying their contents, e.g., a category image could be parameterized with the category to apply (e.g., feature type or building usage) and mask images could be generated for specific features (e.g., specified by GML-id). Using this image context definition, the *WPVS++* `GetView` operation is modified and extended in respect of multiple camera specifications as described in Listing 1.

4 Extending the Interaction Dimension

We consider the support of interaction with the generated image as a main requirement of the *WPVS++*. This includes techniques for maneuvering

[2] Due to possible non-perspective projections, the name of the *WPVS++* should be adapted in future, e.g., to Web View Service.

the virtual camera through the 3D world and for retrieving information about the visualized objects. In this section, we describe extensions of the WPVS that support navigation within and information retrieval from the rendered images.

Listing 1. Extended interface of the *WPVS++*.

```
OP GetView (mand.)
 IN
  ImageContext(1,n)
 OUT
  Image(1,n): Image

OP GetCategory (opt.)
 IN
  CategoryType: string
  ObjectId: integer
 OUT
  Category: string

OP GetPosition (opt.)
 IN
  ImageContext: ImageContext
  Coordinate2D(1,n): Vector2D
 OUT
  Coordinate3D(0,n): Vector3D

OP GetFeatureInfo (opt.)
 IN
  ImageContext: ImageContext
  Coordinate2D(1,n): Vector2D
  ResponseFormat: string
 OUT
  FeatureInfo(0,n): anyType

OP GetNavigationCamera (opt.)
 IN
  ImageContext: ImageContext
  Coordinate2D(1,n): Vector2D
  NavigationType: MoveCamera
  ResponseFormat: string
 OUT
  Camera : Camera

OP GetMeasurementResult (opt.)
 IN
  ImageContext: ImageContext
  Coordinate2D(2,*): Vector2D
  MeasurementType: PATH | AREA
  ResponseFormat: string
 OUT
  MeasurementResult: string

TYPE Camera
 PointOfCamera: Vector3D
 PointOfInterest: Vector3D
 UpVector: decimal (opt.)
 Proj: Perspective|Orthographic
 NearPlane: decimal
 FarPlane: decimal

TYPE Perspective
 FovY: decimal
 FovX: decimal (opt.)

TYPE Orthographic
 Left: decimal
 Right: decimal
 Top: decimal
 Bottom: decimal

TYPE ImageContext
 CRS: string
 Dataset(1,k): Dataset
 BoundingBox(0,1): BoundingBox
 Style(1,k) : string
 SLD(0,1): SLD-Ref
 Camera(1,m): Camera
 Width: integer
 Height: integer
 ImageLayer(1,n): ImageType
 OutputParameter(0,n): string
 OutputFormat(1,n): string

ENUM ImageType
 { COLOR, DEPTH, OBJECTID,
   NORMAL, MASK }
```

As described, feature-related information could be bulk-loaded to a *WPVS++* consumer as non-color layers, e.g., object id layers or category

layers. This is particularly useful, when this data is required for a lot of pixels of the image, as bulk loading avoids the transmission overhead caused by multiple requests. However, if only sparse information is needed, or in the case of a very simple client, an operation for requesting particular feature information at a 2D pixel position would be suitable; transfer load would be reduced and a client would not have to parse and evaluate the whole image layers.

4.1 Requesting 3D Coordinates

Retrieving 3D geocoordinates for a specific 2D pixel position in a generated image is useful for many applications (e.g., for specifying the position of a defect fireplug). For this, we propose a *WPVS++* operation `GetPosition` (Listing 1). As the *WPVS++* is stateless, in addition to the 2D pixel position the image specification of the original `GetView` request has to be part of the `GetPosition` request. This operation is highly useful for navigation in the visualized 3DGeoVE: A user could select two positions in the image that specify the desired point of camera and point of interest; the client requests corresponding 3D coordinates, and uses these for specifying the camera in a subsequent `GetView` request.

4.2 Requesting Feature Information

Similar to the WMS, a `GetFeatureInfo` operation should provide extra information about the geoobjects at a specific pixel position. This operation could either provide domain-specific or any potentially useful information to the consumer. According to the underlying data source and its capabilities, various response formats could be supported, e.g., attribute names and values in XML format or even GML structured data sets. Additional formats can be supported and specified as mime type within the service request. In the case of structured responses (e.g., in GML format), the WPVS operation `GetDescription` can be used for retrieving schema information for specific datasets.

4.3 Navigation based on Picking

Navigation represents a fundamental interaction type for 3DGeoVEs. From virtual globes and online map systems users know to navigate by using mouse and keyboard in a real-time interactive manner. However, especially due to the image-based approach, the WPVS inherently provides

only non-real-time step-by-step navigation. Assistant navigation techniques could compensate that drawback and allow a user to interact with the image, e.g., by sketching a desired point of interest. The portrayal service automatically interprets this navigation input, taking into account scene objects, their types, and their navigation affordances, and computes and responds a camera specification, which can be used for requesting the corresponding view.

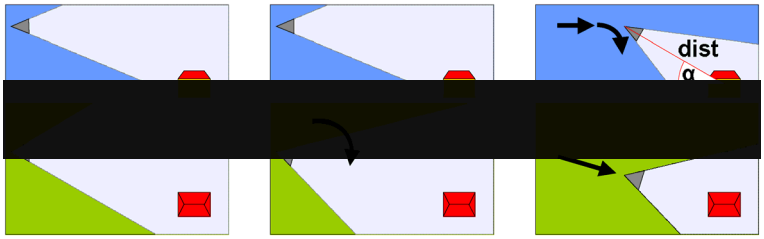

Fig. 3. Changing the camera orientation and position; the camera is oriented to the selected object, moved there while keeping its height, and finally looks down in a specified angle. Side view (top) and top view (bottom).

However, complex, arbitrary, or domain-specific navigation techniques can hardly be defined and described in a formal interface specification. Because of that, we propose an optional generic operation `GetNavigationCamera` that can compute and provide a meaningful camera specification for a 2D pixel position (List. 1). For virtual 3D city models, we suggest a comprehensible navigation technique that supports a user in moving closer to a selected object. This technique moves the camera toward the center of the bounding box of a selected object, while the camera height is kept constant; finally, the camera is positioned in a specified angle above the object (Fig. 3). This ensures that the camera is only moved along a line of sight keeping the probability of occlusion low.

4.4 Measurement based on Picking

Measuring within a requested view represents a main functionality used to retrieve information about the spatial extent of geofeatures, their spatial relationships, and the overall spatial layout of a 3D scene. Hence, *WPVS++* should support the measurement of distances, paths, and areas (List. 1):

- *Path measurement:* Computes the sum of the Euclidean distances between the 3D positions derived from each pair of consecutive 2D pixel

positions. If only two 2D pixel positions are provided as input, the *WPVS++* performs distance measurement.

- *Area measurement:* Computes the area which is outlined by the input points. In 3D, area measurement is not straight forward, as the derived 3D points are not likely to be coplanar. Projecting the 3D points onto a horizontal 2D plane would be a straight forward solution for this problem. More accurate algorithms are possible, e.g., computing a best fitting plane for projection or applying differential techniques to the problem.

5 Implementation

5.1 *WPVS++*

For demonstrating the applicability of the proposed concepts and extensions, we have implemented a prototype of the described *WPVS++* service and tested it with a virtual 3D city model. This implementation is based on the Virtual Rendering System (VRS). The demo dataset is given in 3DS format. Thus, only object names and unique object ids can be requested via `GetFeatureInfo` operation. The service is accessed by an HTTP interface on top of TCP/IP.

For showing the applicability of the *WPVS++*, we have implemented two client applications, which differ in their requirements concerning processing power, rendering capabilities, and degree of interactivity.

5.2 Lightweight JavaScript Client

As example of very lightweight clients, we implemented a JavaScript-based client, which can be run in a web browser without additional plug-ins or libraries. This client just requests and displays color images by HTML `<img/>` elements. It provides step-by-step navigation such as moving left/right, tilting up/down, or rotating around the POC or POI. For each navigation step a new image is requested and displayed.

Moreover, this client takes advantage of the `GetPosition` operation for retrieving 3D coordinates for changing the virtual camera's orientation and for implementing a "move there and orient here" navigation. The `GetFeatureInfo` operation is used for retrieving object information about features in the image.

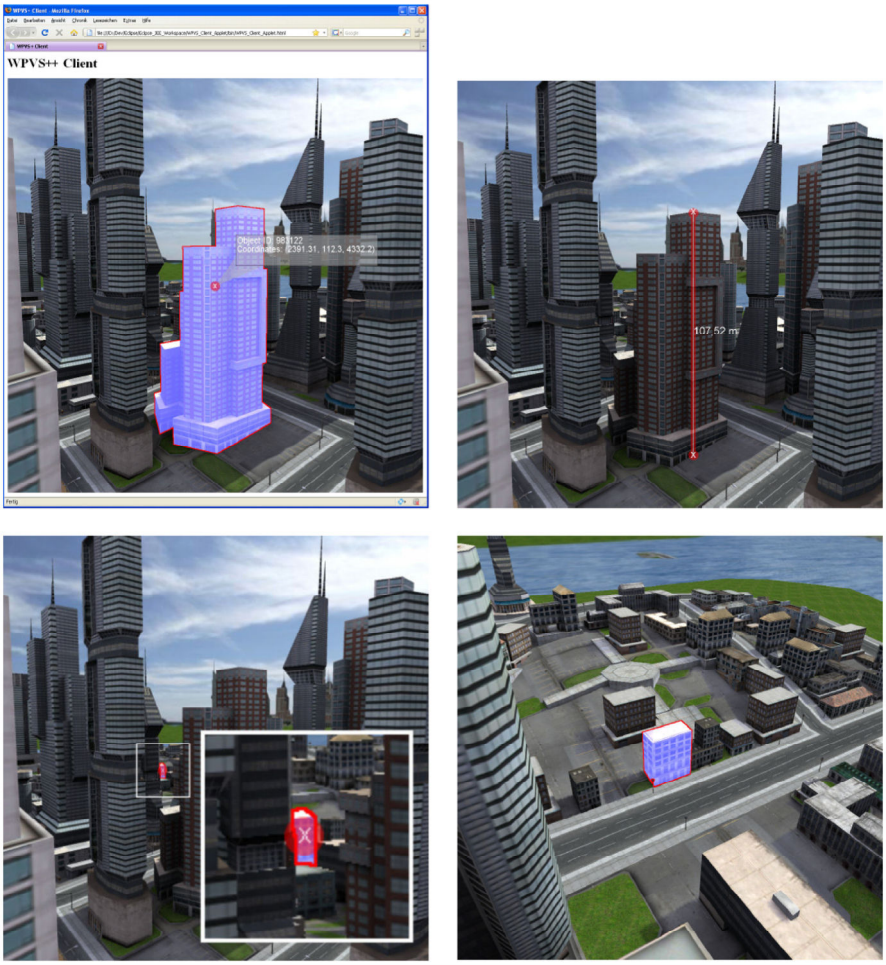

Fig. 4. Java-based interactive web client. Object selection by using the object id layer (upper left); distance measurement by calculating 3D positions from the depth layer (upper right); moving the camera to a selected object by requesting *GetNavigationCamera* (lower left and right).

5.3 Java-based Client

A second client application (Fig. 4) implements functionalities for image processing and rendering; thus, it requires more processing power. It is Java-based and can be integrated into a web page as a Java applet, which also allows for a broad range of applications.

In addition to the color layer, this client also requests object id and depth layers. Object ids are used for selecting objects of the image. When an object is selected, the client highlights all the pixels that belong to this object and emphasizes its contour. Depth data is used for computing the 3D point at each pixel of the image. Using this, the client provides an interactive distance measurement tool. After selecting an image position, the distance from this point to the 3D point under the mouse cursor is computed and displayed as an image overlay.

6 Results and Discussion

6.1 Measuring the *WPVS++* Response Behavior

Our current *WPVS++* implementation provides only a single rendering server. Thus, synchronous requests are processed sequentially. For measuring the service's response behavior, the Java client stressed the *WPVS++* by repeatedly requesting 120 color, depth, and object id layers. Up to 10 requests have been sent in parallel. Furthermore, the performance of multipart responses was tested by requesting 40 image sets (color, depth, and object id layer). The test was performed in an intranet environment; rendering was performed on a desktop PC (2.4GHz double core, 2 GB RAM, NV GeForce 7900 GS), the Java client was executed on a different PC connected to the network.

Table 1 lists the number of images per second that the service can render, compress and send, and the client can decompress. Generally, providing multiple images in one response via HTTP multipart provides higher throughput than requesting each image layer separately. This could be caused by reduced message overhead, i.e., sending and processing the requests. The gain increases for smaller images.

Table 1. Maximum number of processed requests per second when requesting color layer, depth layer, and object id layer separately (single part response) or within a single request (multipart response) using three image sizes.

	256 x 256	512 x 512	1024 x 1024
Single part response	4.7	3.0	1.5
Multipart response	13.9	5.7	2.6
Factor	2.95	1.90	1.73

6.2 Measuring the Size of Image Layers

We logged the size of generated and transferred image layers while navigating through a virtual 3D city model at the Java client. For an overall number of 370 camera positions, the client requested color layers (24 bit JPEG), depth layers (32 bit PNG), and object id and normal layers (24 bit PNG each) in a size of 512 x 512 pixels. The navigation process included overviews as well as pedestrian views. While the depth layer accounts for nearly two-thirds of the transmitted data, object id layer and normal layer play only a minor role (Table 2). This is because for depth layers, only low compression rates can be reached, due to the little pixel-to-pixel coherence. Thus, alternative encodings for mapping depth values to image data are required in order to reach better compression rates.

Table 2. Image layer size in Kbytes when navigating through a virtual 3D city model and percentage of each image layer type.

	Minimum	Maximum	Average	Avg. percentage
COLOR	8.37	126.11	77.95	0.25
DEPTH	5.98	725.22	199.63	0.64
OBJECTID	3.50	19.05	9.56	0.03
NORMAL	3.50	85.95	24.30	0.08

6.3 Comparison to full 3D Geovirtual Environments

Compared to full 3DGeoVEs, the *WPVS++* provides only visual representations. Modalities such as auditory or tactile perception, are not addressed. However, provided a sufficient bandwidth, we assume high usability rates for applications and systems based on the *WPVS++*.

Navigation capabilities mainly depend on the functionality of the clients that consume the *WPVS++*. Simple clients can provide a step-by-step navigation based on requesting and displaying single views. For this, the additional `GetPosition` operation facilitates an easy and targeted camera manipulation. Based on depth information, more complex clients could apply, e.g., image-based rendering techniques for providing more convenient and close to real-time visualization and navigation functionality.

Regarding object interaction, the *WPVS++* supports retrieving information for individual objects of the scene. Object manipulation capabilities are limited: For persistent manipulations of feature geometries and attributes the original data sources must be accessible, e.g., via a Web Feature Service (WFS) [17]. For user-defined appearance manipulation, effective styling mechanisms and specifications are required.

7 Conclusions and Future Work

In this work, we present and discuss the *WPVS++*, an extended version of the OGC Web Perspective View Service (WPVS), which aims at enhanced interaction and navigation capabilities. The *WPVS++* includes functionalities for the exploration and analysis of presented 3DGeoVEs by simple clients, e.g., by requesting 3D positions, retrieving object ids and measuring distances. Additionally, the *WPVS++* provides non-color image layers, which represent thematic information and can be utilized for enhancing client-side interaction. We demonstrate the implementation of major *WPVS++* features with two different client applications and a common application example.

The extended *WPVS++* contributes to making complex and massive 3DGeoVEs accessible and interactively usable. Due to the HTTP-based interface, it can be easily integrated into web-based applications and systems, e.g., for urban planning, public participation systems, location marketing, or threat or emergency response. The additional non-color image layers of the *WPVS++* facilitate advanced image processing capabilities within spatial data infrastructures, e.g., for image-based annotation of the 3DGeoVE or for advanced rendering effects. Within a spatial data infrastructure, the *WPVS++* could serve as frontend to a Web 3D Service (W3DS): The *WPVS++* could request the 3D data as scene graph from W3DS, and render and distribute images.

In future work, we plan to fully implement the proposed functionalities, to further formalize the extension concepts, and to research techniques for further improving the interaction process in terms of efficiency and interactivity. For example, we will investigate how to better compress depth image layers while still using standard encodings. As a further task, we will extend the client application by image-based rendering techniques for better using the potentials of the image-based portrayal service. Additionally, we will investigate the service-based provisioning of non-photorealistic rendering and illustrative depictions, which could support and ease the cognition of 3D views.

References

1. Altmaier A, Kolbe Th (2003) Applications and Solutions for Interoperable 3D Geo-Visualization. In: Fritsch D (ed) Photogrammetric Week. Wichmann
2. Capin T, Pulli K, Akenine-Möller Th (2008) The State of the Art in Mobile Graphics Research. Computer Graphics Application 28(4):74–84

3. Chang C and Ger S (2002) Enhancing 3D Graphics on Mobile Devices by Image-Based Rendering. In: Proc. of the 3rd Pacific Rim Conference on Multimedia (PCM '02), Springer, pp 1105–1111
4. de la Beaujardiere J (2006) OpenGIS Web Map Server Implementation Specification. Version 1.3.0, Open Geospatial Consortium Inc.
5. Faugeras O, Laveau S, Robert L, Csurka G, Zeller C (1995) 3D Reconstruction of Urban Scenes from Sequences of Images. In: Gruen A, Kuebler O, Agouris P (eds) Automatic Extraction of Man-Made Objects from Aerial and Space Images. . Birkhuser, pp 145–168
6. Gu X, Gortler SJ, Hoppe H (2002) Geometry images. ACM Trans. Graph 21(3):355–361
7. Lapidous E, Jiao G (1999) Optimal depth buffer for low-cost graphics hardware. In: Proc. of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware (HWWS '99). New York, ACM, pp 67–73
8. Lorenz H, Trapp M, Jobst M, Döllner J (2008) Interactive Multi-Perspective Views of Virtual 3D Landscape and City Models. In: Proc. of the 11th AGILE Int. Conf. on GI Science. Springer
9. Nadalutti D, Chittaro L, Buttussi F (2006) Rendering of X3D content on mobile devices with OpenGL ES. In: Proc. of the 11th Int. Conf. on 3D web technology (Web3D '06). New York, pp 19–26
10. Nurminen A (2008) Mobile 3D City Maps. In: IEEE Computer Graphics and Applications 28(4):20–31
11. OpenEXR documentation. http://www.openexr.com/documentation.html (14.06.2009).
12. Percivall G (ed) (2008) OGC Reference Model. Version 2.0, Open Geospatial Consortium Inc.
13. Quadt U, Kolbe Th (ed) (2005) Web 3D Service, Version 0.3.0, OGC Discussion Paper, Open Geospatial Consortium Inc.
14. Saito T, Takahashi T (1990) Comprehensible rendering of 3-D shapes. In: SIGGRAPH Computer Graphics 24(4):197–206
15. Singh RR (ed.) (2001) Web Terrain Server (WTS). Version 0.3.2, OGC Discussion Paper, Open Geospatial Consortium Inc.
16. Verlani P, Goswami A, Narayanan PJ, Dwivedi S, Penta SK (2006). Depth Images: Representations and Real-Time Rendering. In: Proc. of Int. Symp. on 3D Data Processing Visualization and Transmission, pp 962–969
17. Vretanos PA (ed) (2005) Web Feature Service Implementation Specification. Version 1.1.0, Open Geospatial Consortium Inc.